

772  
AM  
LANGUAGE RECOGNITION USING SIMILARITY  
~~PATTERN MATCHING METHOD AND APPARATUS~~ MEASURE

The present invention relates to an apparatus and method for matching sequences of phonemes or the like. The invention can be used to search a database of data files having associated phonetic annotations, in response to a user's input query. The input query may be a voiced or typed query.

A27

Databases of information are well known and suffer from the problem of how to locate and retrieve the desired information from the database quickly and efficiently. Existing database search tools allow the user to search the database using typed keywords. Whilst this is quick and efficient, this type of searching is not suitable for various kinds of databases, such as video or audio databases.

A recent proposal has been made to annotate such video and audio databases with a phonetic transcription of the speech content in the audio and video files, with subsequent retrieval being achieved by comparing a phonetic transcription of a user's input query with the phoneme annotation data in the database. The technique proposed for matching the sequences of phonemes firstly defines a set of features in the query, each feature being taken as an overlapping fixed size fragment from the phoneme string, it then identifies the frequency of occurrence of the features in both the query and the annotation and then finally determines a measure of the similarity between the query and the annotation using a cosine measure of these frequencies of occurrences. One advantage of this kind of phoneme comparison technique is

that it can cope with situations where the sequence of words of the query do not exactly match the sequence of words of the annotation. However, it suffers from the problem that it is prone to error especially when the query and the annotation are spoken at different speeds and if there are any deletions of parts of words from the query, but not from the annotation, or vice versa.

A37

The present invention aims to provide an alternative technique for matching sequences of phonemes.

According to one aspect, the present invention provides a feature comparison apparatus comprising: input means for receiving an input signal; recognition processing means for comparing said input signal with stored feature models to identify an input sequence of features in said input signal and confidence data representative of the confidence that the input sequence of features is representative of said input signal; and means for comparing said input sequence of features with a stored sequence of features using predetermined similarity data which defines similarities between different features and using said confidence data, to provide a measure of the similarity between the input sequence of features and the stored sequence of features.

According to another aspect, the present invention provides an apparatus for searching a database comprising a plurality of information entries to identify information to be retrieved therefrom, each of said plurality of information entries having an associated annotation comprising a sequence of features, the apparatus comprising: a feature comparison apparatus as

005507-10500

described above for comparing a query sequence of features obtained from an input query with the features of each annotation to provide a set of comparison results; and means for identifying said information to be retrieved from said database using said comparison results.

ALL

Exemplary embodiments of the present invention will now be described with reference to Figures 1 to 21, in which:

Figure 1 is a schematic block diagram illustrating a user terminal which allows the annotation of a data file with annotation data generated from a typed or voiced input from a user;

Figure 2 is a schematic diagram of phoneme and word lattice annotation data which is generated from a typed input by the user for annotating the data file;

Figure 3 is a schematic diagram of phoneme and word lattice annotation data which is generated from a voiced input by the user for annotating the data file;

Figure 4 is a schematic block diagram of a user's terminal which allows the user to retrieve information from the database by a typed or voice query;

Figure 5a is a flow diagram illustrating part of the flow control of the user terminal shown in Figure 4;

Figure 5b is a flow diagram illustrating the remaining part of the flow control of the user terminal shown in Figure 4;

Figure 6a is a schematic diagram which shows an underlying statistical model which is assumed to have generated both the query and the annotation;

5 Figure 6b is a schematic diagram which shows a first sequence of phonemes representative of a typed input and a second sequence of phonemes representative of a user's voice input, and which illustrates the possibility of there being phoneme insertions and deletions from the user's voice input relative to the typed input;

10 Figure 6c is a schematic diagram which shows a first and second sequence of phonemes, each representative of a voiced input and a third sequence of phonemes representative of a canonical sequence of phonemes corresponding to what was actually said in the corresponding voiced inputs, and which illustrates the possibility of there being phoneme insertions and deletions from the two voiced inputs relative to the corresponding canonical sequence of phonemes;

20 Figure 7 schematically illustrates a search space created by the sequence of annotation phonemes and the sequence of query phonemes together with a start null node and an end null node;

25 Figure 8 is a two dimensional plot with the horizontal axis being provided for the phonemes of the annotation and the vertical axis being provided for the phonemes of the query, and showing a number of lattice points, each corresponding to a possible match between an annotation phoneme and a query phoneme;

Figure 9a schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when the annotation is a typed input and the query is generated from a voiced input;

5

Figure 9b schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when the query is a typed input and when the annotation is a voiced input;

10

Figure 10 schematically illustrates the deletion and decoding probabilities which are stored for an example phoneme;

15

Figure 11 schematically illustrates the dynamic programming constraints employed in a dynamic programming matching process when both the annotation and the query are voiced inputs;

20

Figure 12 is a flow diagram illustrating the main processing steps performed in the dynamic programming matching process;

25

Figure 13 is a flow diagram illustrating the main processing steps employed to begin the dynamic programming process by propagating from a null start node to all possible start points;

30

Figure 14 is a flow diagram illustrating the main processing steps employed to propagate dynamic programming paths from the start points to all possible end points;

005507 105500

Figure 15 is a flow diagram illustrating the main processing steps employed in propagating the paths from the end points to a null end node;

5 Figure 16a is a flow diagram illustrating part of the processing steps performed in propagating a path using the dynamic programming constraints;

10 Figure 16b is a flow diagram illustrating the remaining process steps involved in propagating a path using the dynamic programming constraints;

15 Figure 17 is a flow diagram illustrating the processing steps involved in determining a transition score for propagating a path from a start point to an end point;

20 Figure 18a is a flow diagram illustrating part of the processing steps employed in calculating scores for deletions and decodings of annotation and query phonemes;

Figure 18b is a flow diagram illustrating the remaining steps employed in determining scores for deletions and decodings of annotation and query phonemes;

25 Figure 19a schematically illustrates an alternative embodiment which employs a different technique for aligning the query with each annotation;

30 Figure 19b is a plot illustrating the way in which a dynamic programming score varies with a comparison of a query with an annotation in the embodiment illustrated in Figure 19a;

005207 4050960

Figure 20 is a schematic block diagram illustrating the form of an alternative user terminal which is operable to retrieve a data file from a database located within a remote server in response to an input voice query; and

Figure 21 illustrates another user terminal which allows a user to retrieve data from a database located within a remote server in response to an input voice query.

A57

Embodiments of the present invention can be implemented using dedicated hardware circuits, but the embodiment to be described is implemented in computer software or code, which is run in conjunction with processing hardware such as a personal computer, workstation, photocopier, facsimile machine, personal digital assistant (PDA) or the like.

#### DATA FILE ANNOTATION

Figure 1 illustrates the form of a user terminal 59 which allows a user to input typed or voiced annotation data via the keyboard 3 and microphone 7 for annotating a data file 91 which is to be stored in a database 29. In this embodiment, the data file 91 comprises a two dimensional image generated by, for example, a camera. The user terminal 59 allows the user 39 to annotate the 2D image with an appropriate annotation which can be used subsequently for retrieving the 2D image from the database 29. In this embodiment, a typed input is converted, by the phonetic transcription unit 75, into phoneme (or phoneme-like) and word lattice annotation data which is passed to the control unit 55. Figure 2 illustrates the form of the phoneme and word lattice annotation data generated for the typed input "picture of

the Taj Mahal". As shown in Figure 2, the phoneme and word lattice is an acyclic directed graph with a single entry point and a single exit point. It represents different parses of the user's input. As shown, the phonetic transcription unit 75 identifies a number of different possible phoneme strings which correspond to the typed input, from an internal phonetic dictionary (not shown).

Similarly, a voiced input is converted by the automatic speech recognition unit 51 into phoneme (or phoneme-like) and word lattice annotation data which is also passed to the control unit 55. The automatic speech recognition unit 51 generates this phoneme and word lattice annotation data by (i) generating a phoneme lattice for the input utterance; (ii) then identifying words within the phoneme lattice; and (iii) finally by combining the two. Figure 3 illustrates the form of the phoneme and word lattice annotation data generated for the input utterance "picture of the Taj Mahal". As shown, the automatic speech recognition unit identifies a number of different possible phoneme strings which correspond to this input utterance. As is well known in the art of speech recognition, these different possibilities can have their own weighting which is generated by the speech recognition unit 51 and is indicative of the confidence of the speech recognition unit's output. In this embodiment, however, this weighting of the phonemes is not performed. As shown in Figure 3, the words which the automatic speech recognition unit 51 identifies within the phoneme lattice are incorporated into the phoneme lattice data structure. As shown for the example phrase given above, the automatic speech recognition unit 51

005507 102500



identifies the words "picture", "of", "off", "the", "other", "ta", "tar", "jam", "ah", "hal", "ha" and "al".

As shown in Figure 3, the phoneme and word lattice generated by the automatic speech recognition unit 51 is an acyclic directed graph with a single entry point and a single exit point. It represents different parses of the user's input annotation utterance. It is not simply a sequence of words with alternatives, since each word does not have to be replaced by a single alternative, one word can be substituted for two or more words or phonemes, and the whole structure can form a substitution for one or more words or phonemes. Therefore, the density of the data within the phoneme and word lattice annotation data essentially remains linear throughout the annotation data, rather than growing exponentially as in the case of a system which generates the N-best word lists for the audio annotation input.

In this embodiment, the annotation data generated by the automatic speech recognition unit 51 or the phonetic transcription unit 75 has the following general form:

#### HEADER

- flag if word if phoneme if mixed
- time index associating the location of blocks of annotation data within memory to a given time point.
- word set used (i.e. the dictionary)
- phoneme set used
- the language to which the vocabulary pertains
- phoneme probability data

Block(i) i = 0,1,2,.....

00505077 102500

node  $N_j$      $j = 0, 1, 2, \dots$

- time offset of node from start of block
- phoneme links (k)  $k = 0, 1, 2, \dots$

offset to node  $N_j = N_k - N_j$  ( $N_k$  is node to which link  $k$  extends)

phoneme associated with link (k)

- word links (l)  $l = 0, 1, 2, \dots$

offset to node  $N_j = N_l - N_j$  ( $N_l$  is node to which link  $l$  extends)

word associated with link (l)

The flag identifying if the annotation data is word annotation data, phoneme annotation data or if it is mixed is provided since not all the data files within the database will include the combined phoneme and word lattice annotation data discussed above, and in this case, a different search strategy would be used to search this annotation data.

In this embodiment, the annotation data is divided into blocks of nodes in order to allow the search to jump into the middle of the annotation data for a given search. The header therefore includes a time index which associates the location of the blocks of annotation data within the memory to a given time offset between the time of start and the time corresponding to the beginning of the block.

The header also includes data defining the word set used (i.e. the dictionary), the phoneme set used and their probabilities and the language to which the vocabulary pertains. The header may also include details of the automatic speech recognition system used to generate the

005507 205500

annotation data and any appropriate settings thereof which were used during the generation of the annotation data.

5 The blocks of annotation data then follow the header and identify, for each node in the block, the time offset of the node from the start of the block, the phoneme links which connect that node to other nodes by phonemes and word links which connect that node to other nodes by words. Each phoneme link and word link identifies the phoneme or word which is associated with the link. They also identify the offset to the current node. For example, if node  $N_{50}$  is linked to node  $N_{55}$  by a phoneme link, then the offset to node  $N_{50}$  is 5. As those skilled in the art will appreciate, using an offset indication like this allows the division of the continuous annotation data into separate blocks.

10 In an embodiment where an automatic speech recognition unit outputs weightings indicative of the confidence of the speech recognition units output, these weightings or confidence scores would also be included within the data structure. In particular, a confidence score would be provided for each node which is indicative of the confidence of arriving at the node and each of the phoneme and word links would include a transition score depending upon the weighting given to the corresponding phoneme or word. These weightings would then be used to control the search and retrieval of the data files by discarding those matches which have a low confidence score.

In response to the user's input, the control unit 55

005207 2056960

retrieves the appropriate 2D file from the database 29 and appends the generated phoneme and word annotation data to the data file 91. The augmented data file is then returned to the database 29. During this annotating step, the control unit 55 is operable to display the 2D image on the display 57, so that the user can ensure that the annotation data is associated with the correct data file 91.

As will be explained in more detail below, the use of such phoneme and word lattice annotation data allows a quick and efficient search of the database 29 to be carried out, to identify and to retrieve a desired 2D image data file stored therein. This can be achieved by firstly searching in the database 29 using the word data and, if this search fails to provide the required data file, then performing a further search using the more robust phoneme data. As those skilled in the art of speech recognition will realise, use of phoneme data is more robust because phonemes are dictionary independent and allow the system to cope with out of vocabulary words, such as names, places, foreign words etc. Use of phoneme data is also capable of making the system future-proof, since it allows data files which are placed into the database 29 to be retrieved when the original annotation was input by voice and the original automatic speech recognition system did not understand the words of the input annotation.

#### DATA FILE RETRIEVAL

Figure 4 is a block diagram illustrating the form of a user terminal 59 which is used, in this embodiment, to retrieve the annotated 2D images from the database 29.

00595072-102500

005077 102500

This user terminal 59 may be, for example, a personal computer, a hand-held device or the like. As shown, in this embodiment, the user terminal 59 comprises the database 29 of annotated 2D images, an automatic speech recognition unit 51, a phonetic transcription unit 75, a keyboard 3, a microphone 7, a search engine 53, a control unit 55 and a display 57. In operation, the user inputs either a voice query via the microphone 7 or a typed query via the keyboard 3 and the query is processed either by the automatic speech recognition unit 51 or the phonetic transcription unit 75 to generate corresponding phoneme and word data. This data may also take the form of a phoneme and word lattice, but this is not essential. This phoneme and word data is then input to the control unit 55 which is operable to initiate an appropriate search of the database 29 using the search engine 53. The results of the search, generated by the search engine 53, are then transmitted back to the control unit 55 which analyses the search results and generates and displays appropriate display data (such as the retrieved 2D image) to the user via the display 57.

Figures 5a and 5b are flow diagrams which illustrate the way in which the user terminal 59 operates in this embodiment. In step s1, the user terminal 59 is in an idle state and awaits an input query from the user 39. Upon receipt of an input query, the phoneme and word data for the input query is generated in step s3 by the automatic speech recognition unit 51 or the phonetic transcription unit 75. The control unit 55 then instructs the search engine 53, in step s5, to perform a search in the database 29 using the word data generated from the input query. The word search employed in this

embodiment is the same as is currently being used in the art for typed word searches, and will not be described in more detail here. If in step s7, the control unit 55 identifies from the search results, that a match for the user's input query has been found, then it outputs the search results to the user via the display 57.

In this embodiment, the user terminal 59 then allows the user to consider the search results and awaits the user's confirmation as to whether or not the results correspond to the information the user requires. If they are, then the processing proceeds from step s11 to the end of the processing and the user terminal 59 returns to its idle state and awaits the next input query. If, however, the user indicates (by, for example, inputting an appropriate voice command) that the search results do not correspond to the desired information, then the processing proceeds from step s11 to step s13, where the search engine 53 performs a phoneme search of the database 29. However, in this embodiment, the phoneme search performed in step s13 is not of the whole database 29, since this could take several hours depending on the size of the database 29.

Instead, the phoneme search performed in step s13 uses the results of the word search performed in step s5 to identify one or more portions within the database which may correspond to the user's input query. For example, if the query comprises three words and the word search only identifies one or two of the query words in the annotation, then it performs a phoneme search of the portions of the annotations surrounding the identified word or words. The way in which the phoneme search

performed in step s13 is carried out in this embodiment will be described in more detail later.

After the phoneme search has been performed, the control unit 55 identifies, in step s15, if a match has been found. If a match has been found, then the processing proceeds to step s17 where the control unit 55 causes the search results to be displayed to the user on the display 57. Again, the system then awaits the user's confirmation as to whether or not the search results correspond to the desired information. If the results are correct, then the processing passes from step s19 to the end and the user terminal 59 returns to its idle state and awaits the next input query. If however, the user indicates that the search results do not correspond to the desired information, then the processing proceeds from step s19 to step s21, where the control unit 55 is operable to ask the user, via the display 57, whether or not a phoneme search should be performed of the whole database 29. If in response to this query, the user indicates that such a search should be performed, then the processing proceeds to step s23 where the search engine performs a phoneme search of the entire database 29.

On completion of this search, the control unit 55 identifies, in step s25, whether or not a match for the user's input query has been found. If a match is found, then the processing proceeds to step s27 where the control unit 55 causes the search results to be displayed to the user on the display 57. If the search results are correct, then the processing proceeds from step s29 to the end of the processing and the user terminal 59

returns to its idle state and awaits the next input query. If, on the other hand, the user indicates that the search results still do not correspond to the desired information, then the processing passes to step s31 where the control unit 55 queries the user, via the display 57, whether or not the user wishes to redefine or amend the search query. If the user does wish to redefine or amend the search query, then the processing returns to step s3 where the user's subsequent input query is processed in a similar manner. If the search is not to be redefined or amended, then the search results and the user's initial input query are discarded and the user terminal 59 returns to its idle state and awaits the next input query.

A general description has been given above of the way in which a search is carried out in this embodiment by the user terminal 59. A more detailed description will now be given of the way in which the search engine 53 carries out the phoneme searches, together with a brief description of the motivation underlying the search strategy.

#### INFORMATION RETRIEVAL AS A CLASSIFICATION PROBLEM

In the classic classification scenario, a test datum has to be classified into one of  $K$  classes. This is done using knowledge about other data for which the class is known. The classification problem assumes that there is a "class" random variable which can take values from 1 to  $K$ . The optimal classification then being found by identifying the class to which the test datum most likely belongs. It is assumed that the training data is generated by  $N$  generative processes which resulted in  $n_k$

005207-105500



data of class  $k$ , where  $\sum_{k=1}^K n_k = N$ . Denoting the vector  $(n_1, n_2, \dots, n_K)$  by  $n$ , the training data by  $D$  and the test datum by  $x$ , the classic classification problem is to determine the value of  $k$  which maximises the following probability:

$$P(k|x, D, n) = \frac{P(x|k, D, n)P(k|D, n)}{P(x|D)} \quad (1)$$

The second term on the numerator is a prior probability for the class which gives more weight to classes which occur more often. In the context of information retrieval, each class has a single training datum (i.e. the annotation data). Therefore, for information retrieval, the second term on the right-hand side of the above expression can be ignored. Similarly, the denominator can also be ignored since  $P(x|D)$  is the same for each class and therefore just normalises the numerator. Consequently, the order of the classes can be ranked by simply ranking the order of the first term on the numerator of the above expression for the classes. In other words, determining and ranking  $P(x|d_k)$  for all the classes, where  $d_k$  is the training datum for class  $k$ .

In this embodiment, the test datum  $x$  represents the input query and the training datum for class  $k$  (i.e.  $d_k$ ) represents the  $k^{\text{th}}$  annotation, and it is assumed that there is an underlying statistical model ( $M$ ) which generated both the query and the annotation, as illustrated in Figure 6a. In the general case, this model has three unknowns: the model structure,  $m$ , the state sequences through that model for both the query and

005201 2056960

the annotation,  $s_q$  and  $s_a$ , and the output distribution  $C$ . In this case, we know the output distribution since it embodies the characteristics of the speech recognition system which generates the phoneme strings from the input speech. As will be described later, it can be obtained by applying a large database of known speech to the speech recognition system, and it will be referred to hereinafter as the confusion statistics. Therefore, introducing the state sequences and the model into the above probabilities (and using the variables  $q$  for the input query and  $a$  for the annotation) yields:

$$P(q|a) = \sum_m \sum_{s_a} \sum_{s_q} P(q|m, s_q, s_a, C, a) P(m, s_q, s_a | C, a) \quad (2)$$

which can be expanded using Bayesian methods to give:

$$P(q|a) = \frac{\sum_m \sum_{s_a} \sum_{s_q} P(q|m, s_q, C) P(a|m, s_a, C) P(s_q|m, C) P(s_a|m, C) P(m|C)}{\sum_m \sum_{s_a} \sum_{s_q} P(a|m, s_a, C) P(s_a|m, C) P(m|C)} \quad (3)$$

Although the above expression looks complicated, the summations over the set of state sequences  $s_q$  and  $s_a$  can be performed using a standard dynamic programming algorithm. Further, the last term on both the numerator

and the denominator can be ignored, since it can be assumed that each model is equally likely and the state sequence terms  $P(s|m,c)$  can be ignored because it can also be assumed that each state sequence is equally likely. Further, by assuming that the underlying model structure is a canonical sequence of phonemes having approximately the same length as the query, subject to insertions, the summation over the different models can be removed, although it is replaced with a summation over all possible phonemes because, in the general case, the canonical sequence of phonemes of the model is unknown. Therefore, ignoring the state sequence summations, the term which is to be evaluated inside the dynamic programming algorithm becomes:

$$\sum_{r=1}^{N_p} P(a_i|p_r C)P(q_j|p_r C)P(p_r|C) \quad (4)$$

on the numerator, and

$$\sum_{r=1}^{N_p} P(a_i|p_r C)P(p_r|C) \quad (5)$$

on the denominator (i.e. the normalising term), where  $N_p$  is the total number of phonemes known to the system and  $a_i$ ,  $q_j$  and  $p_r$  are the annotation phoneme, query phoneme and model phoneme respectively which correspond to the current DP lattice point being evaluated. As can be seen from a comparison of equations (4) and (5), the probability terms calculated on the denominator are calculated on the numerator as well. Therefore, both

terms can be accumulated during the same dynamic programming routine. Considering the probabilities which are determined in more detail,  $P(q_j|p_r, C)$  is the probability of decoding canonical phoneme  $p_r$  as query phoneme  $q_j$  given the confusion statistics;  $P(a_i|p_r, C)$  is the probability of decoding canonical phoneme  $p_r$  as annotation phoneme  $a_i$  given the confusion statistics; and  $P(p_r|C)$  is the probability of canonical phoneme  $p_r$  occurring unconditionally given the confusion statistics.

In addition to the above terms, at each point in the dynamic programming calculation, a further term must be calculated which deals with insertions and deletions in the query or the annotation relative to the model. As those skilled in the art will appreciate, an insertion or a deletion in the query is independent from an insertion or a deletion in the annotation and vice versa. Therefore, these additional terms are dealt with separately. Insertions and deletions in the annotation relative to the model must also be considered for the normalising term given in equation (5) above.

As those skilled in the art will appreciate from the description of Figures 4 and 5, in this embodiment, the annotation phoneme data and the query phoneme data may both be derived either from text or speech. Therefore, there are four situations to consider:

- i) both the annotation and the query are generated from text;
- ii) the annotation is generated from text and the query is generated from speech;
- iii) the annotation is generated from speech and the

- query is generated from text; and
- iv) both the query and annotation are generated from speech.

5 The first situation is the simple case in which there can be no time compression/expansion of the annotation or the query and the comparison between the annotation and the query is performed by a simple boolean comparison of the respective phoneme sequences.

10

In the second situation, the annotation is taken to be correct and the dynamic programming alignment allows the insertion and deletion of phonemes in the query in order to find the best alignment between the two. To

15

illustrate this case, Figure 6b shows a possible matching between a sequence of annotation phonemes (labelled  $a_0, a_1, a_2 \dots$ ) and a sequence of query phonemes (labelled  $q_0, q_1, q_2 \dots$ ), when the annotation phonemes are generated from text. As illustrated by the dashed arrows,

20

annotation phoneme  $a_0$  is aligned with query phoneme  $q_0$ , annotation phoneme  $a_1$  is aligned with query phoneme  $q_2$ , annotation phoneme  $a_2$  is aligned with query phoneme  $q_3$ , annotation phoneme  $a_3$  is aligned with query phoneme  $q_3$  and annotation phoneme  $a_4$  is aligned with query phoneme  $q_4$ . For each of these alignments, the dynamic

25

programming routine calculates the terms given in equations (4) and (5) above. However, in this case, these equations simplify because the canonical sequence of model phonemes is known (since these are the annotation phonemes). In particular, the normalising term is one because the annotation is the model and the numerator simplifies to  $P(q_i|a_j, C)$ . In addition to these decoding terms, the dynamic programming routine also

30

005507 102500

calculates the relevant insertion and deletion probabilities for the phonemes which are inserted in the query relative to the annotation (such as query phoneme  $q_1$ ) and for the phonemes which are deleted in the query relative to the annotation (represented by query phoneme  $q_3$  which is matched with the two annotation phonemes  $a_2$  and  $a_3$ ).

The third situation mentioned above is similar to the second situation except the sequence of query phonemes is taken to be correct and the dynamic programming alignment allows the insertion and deletion of phonemes in the annotation relative to the query. However, in this situation, equations (1) to (5) cannot be used because the query is known. Therefore, in this situation, equation (1) can be reformulated as:

$$P(K|x, D, n) = \frac{P(D|K, x, n)P(K|x, n)}{P(D|x)} \quad (6)$$

As with the corresponding terms in the equation (1) above, the second term on the numerator and the denominator can both be ignored. The first term of the numerator in equation (6) above can be expanded in a similar way to the way in which the first term on the numerator of equation (1) was expanded. However, in this situation, with the query being taken to be the model, the normalising term calculated during the dynamic programming routine simplifies to one and the numerator simplifies to  $P(a_i|q_j, C)$ . Like the second situation discussed above, the dynamic programming routine also calculates the relevant insertion and deletion

005507 10500

probabilities for the phonemes which are inserted in the annotation relative to the query and for the phonemes which are deleted in the annotation relative to the query.

5

Finally, in the fourth situation, when both the annotation and the query are generated from speech, both sequences of phoneme data can have insertions and deletions relative to the unknown canonical sequence of model phonemes which represents the text of what was actually spoken. This is illustrated in Figure 6c, which shows a possible matching between a sequence of annotation phonemes (labelled  $a_1, a_{1+1}, a_{1+2} \dots$ ), a sequence of query phonemes (labelled  $q_j, q_{j+1}, q_{j+2} \dots$ ) and a sequence of phonemes (labelled  $p_n, p_{n+1}, p_{n+2} \dots$ ) which represents the canonical sequence of phonemes of what was actually spoken by both the query and the annotation. As shown in Figure 6c, in this case, the dynamic programming alignment technique must allow for the insertion of phonemes in both the annotation and the query (represented by the inserted phonemes  $a_{1+3}$  and  $q_{j+1}$ ) as well as the deletion of phonemes from both the annotation and the query (represented by phonemes  $a_{1+1}$  and  $q_{j+2}$ , which are both aligned with two phonemes in the canonical sequence of phonemes), relative to the canonical sequence of model phonemes.

10

15

20

25

30

As those skilled in the art will appreciate, by introducing the model sequence of phonemes into the calculations, the algorithm is more flexible to pronunciation variations in both the query and the annotation.

005207 42056960

A general description has been given above of the way in which the present embodiment performs information retrieval by matching the sequence of query phonemes with the sequences of annotation phonemes in the database. In order to understand the operation of the present embodiment further, a brief description will now be given of a standard dynamic programming algorithm followed by a more detailed description of the particular algorithm used in this embodiment.

#### OVERVIEW OF DP SEARCH

As those skilled in the art know, dynamic programming is a technique which can be used to find the optimum alignment between sequences of features, which in this embodiment are phonemes. It does this by simultaneously propagating a plurality of dynamic programming paths, each of which represents a possible matching between a sequence of annotation phonemes and a sequence of query phonemes. All paths begin at a start null node, which is at the beginning of the annotation and the query, and propagate until they reach an end null node, which is at the end of the annotation and the query. Figures 7 and 8 schematically illustrate the matching which is performed and this path propagation. In particular, Figure 7 shows a rectangular coordinate plot with the horizontal axis being provided for the annotation and the vertical axis being provided for the query. The start null node  $\emptyset_s$  is provided in the top left hand corner and the end null node  $\emptyset_e$  is provided on the bottom right hand corner. As shown in Figure 8, the phonemes of the annotation are provided along the horizontal axis and the phonemes of the query are provided along the vertical axis. Figure 8 also shows a number of lattice points,

005507 405500



each of which represents a possible alignment between a phoneme of the annotation and a phoneme of the query. For example, lattice point 21 represents a possible alignment between annotation phoneme  $a_3$  and query phoneme  $q_1$ . Figure 8 also shows three dynamic programming paths  $m_1$ ,  $m_2$  and  $m_3$  which represent three possible matchings between the sequences of phonemes representative of the annotation and of the query and which begin at the start null node  $\emptyset_s$  and propagate through the lattice points to the end null node  $\emptyset_e$ . Referring back to equations (2) and (3) above, these dynamic programming paths represent the different state sequences  $s_q$  and  $s_a$  discussed above.

As represented by the different lengths of the horizontal and vertical axes shown in Figure 7, the input query does not need to include all the words of the annotation. For example, if the annotation is "picture of the Taj Mahal", then the user can simply search the database 29 for this picture by inputting the query "Taj Mahal". In this situation, the optimum alignment path would pass along the top horizontal axis until the query started to match the annotation. It would then start to pass through the lattice points to the lower horizontal axis and would end at the end node. This is illustrated in Figure 7 by the path 23. However, as those skilled in the art will appreciate, the words in the query must be in the same order as they appear in the annotation, otherwise the dynamic programming alignment will not work.

In order to determine the similarity between the sequence of annotation phonemes and the sequence of query phonemes, the dynamic programming process keeps a score for each of the dynamic programming paths which it

005207 102500

propagates, which score is dependent upon the overall similarity of the phonemes which are aligned along the path. In order to limit the number of deletions and insertions of phonemes in the sequences being matched, the dynamic programming process places certain constraints on the way in which the dynamic programming paths can propagate. As those skilled in the art will appreciate, these dynamic programming constraints will be different for the four situations discussed above.

#### DP CONSTRAINTS

*Both annotation and query are text.*

In the case where the query phoneme data and the annotation phoneme data are both generated from text, the dynamic programming alignment degenerates into a boolean match between the two phoneme sequences and no phoneme deletions or insertions are allowed.

*Annotation is text and query is speech.*

In the case where the annotation phoneme data is generated from text and the query phoneme data is generated from speech, there can be no phoneme deletions or insertions in the annotation but there can be phoneme deletions and insertions in the query relative to the annotation. Figure 9a illustrates the dynamic programming constraints which are used in this embodiment, when the annotation is generated from text and the query is generated from speech. As shown, if a dynamic programming path ends at lattice point  $(i, j)$ , representing an alignment between annotation phoneme  $a_i$  and query phoneme  $q_j$ , then that dynamic programming path can propagate to the lattice points  $(i+1, j)$ ,  $(i+1, j+1)$  and  $(i+1, j+2)$ . A propagation to point  $(i+1, j)$  represents

the case when there is a deletion of a phoneme from the spoken query as compared with the typed annotation; a propagation to the point  $(i+1, j+1)$  represents the situation when there is a simple decoding between the next phoneme in the annotation and the next phoneme in the query; and a propagation to the point  $(i+1, j+2)$  represents the situation when there is an insertion of phoneme  $q_{j+1}$  in the spoken query as compared with the typed annotation and when there is a decoding between annotation phoneme  $a_{i+1}$  and query phoneme  $q_{j+2}$ .

*Annotation is speech and query is text.*

In the case where the annotation is generated from speech and the query is generated from text, there can be no insertions or deletions of phonemes from the query but there can be insertions and deletions from the annotation relative to the query. Figure 9b illustrates the dynamic programming constraints which are used in this embodiment, when the annotation is generated from speech and the query is generated from text. As shown, if a dynamic programming path ends at lattice point  $(i, j)$ , representing an alignment between annotation phoneme  $a_i$  and query phoneme  $q_j$ , then that dynamic programming path can propagate to the lattice points  $(i, j+1)$ ,  $(i+1, j+1)$  and  $(i+2, j+1)$ . A propagation to point  $(i, j+1)$  represents the case when there is a deletion of a phoneme from the spoken annotation as compared with the typed query; a propagation to the point  $(i+1, j+1)$  represents the situation when there is a simple decoding between the next phoneme in the annotation and the next phoneme in the query; and a propagation to the point  $(i+2, j+1)$  represents the situation when there is an insertion of phoneme  $a_{i+1}$  in the spoken annotation as compared with the

typed query and when there is a decoding between annotation phoneme  $a_{i+2}$  and query phoneme  $q_{i+1}$ .

*Annotation is speech and query is speech.*

5 In the case where both the annotation and the query are generated from speech, phonemes can be inserted and deleted from each of the annotation and the query relative to the other. Figure 11 shows the dynamic programming constraints which are used in this embodiment, when both the annotation phonemes and the query phonemes are generated from speech. In particular, if a dynamic programming path ends at lattice point  $(i,j)$ , representing an alignment between annotation phoneme  $a_i$  and query phoneme  $q_j$ , then that dynamic programming path can propagate to the lattice points  $(i+1,j)$ ,  $(i+2,j)$ ,  $(i+3,j)$ ,  $(i,j+1)$ ,  $(i+1,j+1)$ ,  $(i+2,j+1)$ ,  $(i,j+2)$ ,  $(i+1,j+2)$  and  $(i,j+3)$ . These propagations therefore allow the insertion and deletion of phonemes in both the annotation and the query relative to the unknown canonical sequence of model phonemes corresponding to the text of what was actually spoken.

#### *Beginning and End DP Constraints*

25 In this embodiment, the dynamic programming alignment operation allows a dynamic programming path to start and end at any of the annotation phonemes. As a result, the query does not need to include all the words of the annotation, although the query words do need to be in the same order as they appear in the annotation.

#### DP SCORE PROPAGATION

30 As mentioned above, the dynamic programming process keeps a score for each of the dynamic programming paths, which

005507 205550

score is dependent upon the similarity of the phonemes which are aligned along the path. Therefore, when propagating a path ending at point (i,j) to these other points, the dynamic programming process adds the  
5        respective "cost" of doing so to the cumulative score for the path ending at point (i,j), which is stored in a store (SCORE(i,j)) associated with that point. As those skilled in the art will appreciate, this cost includes the above-mentioned insertion probabilities, deletion  
10        probabilities and decoding probabilities. In particular, when there is an insertion, the cumulative score is multiplied by the probability of inserting the given phoneme; when there is a deletion, the cumulative score is multiplied by the probability of deleting the phoneme;  
15        and when there is a decoding, the cumulative score is multiplied by the probability of decoding the two phonemes.

In order to be able to calculate these probabilities, the  
20        system stores a probability for all possible phoneme combinations. In this embodiment, the deletion of a phoneme in the annotation or the query is treated in a similar manner to a decoding. This is achieved by simply treating a deletion as another phoneme. Therefore, if  
25        there are 43 phonemes known to the system, then the system will store one thousand eight hundred and ninety two ( $1892 = 43 \times 44$ ) decoding/deletion probabilities, one for each possible phoneme decoding and deletion. This is illustrated in Figure 10, which shows the possible  
30        phoneme decodings which are stored for the phoneme /ax/ and which includes the deletion phoneme ( $\emptyset$ ) as one of the possibilities. As those skilled in the art will appreciate, all the decoding probabilities for a given

005507-10500

phoneme must sum to one, since there are no other possibilities. In addition to these decoding/deletion probabilities, the system stores 43 insertion probabilities, one for each possible phoneme insertion. As will be described later, these probabilities are determined in advance from training data.

To illustrate the score propagations, a number of examples will now be considered. In the case where the annotation is text and the query is speech and for the path propagating from point  $(i,j)$  to point  $(i+1,j+2)$ , the phoneme  $q_{j+1}$  is inserted relative to the annotation and query phoneme  $q_{j+2}$  is decoded with annotation phoneme  $a_{i+1}$ . Therefore, the score propagated to point  $(i+1,j+2)$  is given by:

$$S(i+1,j+2) = S(i,j) \cdot PI(q_{j+1}|C) \cdot P(q_{j+2}|a_{i+1},C)$$

(7)

where  $PI(q_{j+1}|C)$  is the probability of inserting phoneme  $q_{j+1}$  in the spoken query and  $P(q_{j+2}|a_{i+1},C)$  represents the probability of decoding annotation phoneme  $a_{i+1}$  as query phoneme  $q_{j+2}$ .

In the case where both the annotation and the query are generated from speech and when propagating from point  $(i,j)$  to point  $(i+2,j+1)$ , the annotation phoneme  $a_{i+1}$  is inserted relative to the query and there is a decoding between annotation phoneme  $a_{i+2}$  and query phoneme  $q_{j+1}$ . Therefore, the score propagated to point  $(i+2,j+1)$  is given by:

$$S(i+2,j+1) = S(i,j) \cdot P(a_{i+1}|C) \cdot \sum_{r=1}^{N_p} P(a_{i+2}|p_r C) P(q_{j+1}|p_r C) P(p_r|C)$$

(8)

As those skilled in the art will appreciate, during this path propagation, several paths will meet at the same lattice point. In this embodiment, the scores associated with the paths which meet are simply added together. Alternatively, a comparison between the scores could be made and the path having the best score could be continued whilst the other path(s) is (are) discarded. However, this is not essential in this embodiment, since the dynamic programming process is only interested in finding a score which represents the similarity between the phoneme data of the query and the phoneme data of the annotation. It is not interested in knowing what the best alignment between the two is.

If both the query and the annotation are generated from speech, then once all the paths have been propagated to the end node  $\emptyset_e$  and a total score for the similarity between the query and the current annotation has been determined, the system normalises this score using the normalising term which has been accumulating during the DP process. The system then compares the query with the next annotation in a similar manner. Once the query has been matched with all the annotations, the normalised scores for the annotations are ranked and based on the ranking, the system outputs to the user the annotation(s) most similar to the input query.

005507 10500

DETAILED DESCRIPTION OF DP SEARCH

A more detailed description will now be given of the way in which the dynamic programming search is carried out when matching a sequence of query phonemes with a sequence of annotation phonemes. Referring to Figure 12, in step s101, the system initialises the dynamic programming scores. Then in step s103, the system propagates paths from the null start node ( $\emptyset_s$ ) to all possible start points. Then in step s105, the system propagates the dynamic programming paths from all the start points to all possible end points using the dynamic programming constraints discussed above. Finally in step s107, the system propagates the paths ending at the end points to the null end node ( $\emptyset_e$ ).

Figure 13 shows in more detail, the processing steps involved in step s103 in propagating the dynamic programming paths from the null start node ( $\emptyset_s$ ) to all the possible start points, which are defined by the dynamic programming constraints. One of the constraints is that a dynamic programming path can begin at any of the annotation phonemes and the other constraint, which defines the number of hops allowed in the sequence of query phonemes, depends upon whether or not the query is text or speech. In particular, if the query is generated from text, then the start points comprise the first row of lattice points in the search space, i.e. points  $(i,0)$  for  $i = 0$  to  $N_{ann}-1$ ; and if the query is generated from speech, then the start points comprise the first four rows of lattice points in the search space, i.e. points  $(i,0)$ ,  $(i,1)$ ,  $(i,2)$  and  $(i,3)$  for  $i = 0$  to  $N_{ann}-1$ .

The way in which this is achieved will now be described

005507 10500  
005507 10500



with reference to the steps shown in Figure 13. As shown, in step s111, the system determines whether or not the input query is a text query. If it is, then the processing proceeds to step s113 where the system sets the value of the variable mx to one, which defines the maximum number of "hops" allowed in the sequence of query phonemes when the query is text. The processing then proceeds to steps s115, s117 and s119 which are operable to start a dynamic programming path at each of the lattice points in the first row of the search space, by adding the transition score for passing from the null start node to the lattice point (i,0) to the score (SCORE(i,0)) associated with point (i,0), for i = 0 to Nann-1. When the query is text, this ends the processing in step s103 shown in Figure 12 and the processing then proceeds to step s105.

If the system determines at step s111, that the query is not text and was therefore generated from a spoken input, the system proceeds to step s121 where mx is set to mxhops which is a constant having a value which is one more than the maximum number of "hops" allowed by the dynamic programming constraints. As shown in Figures 9 and 10, in the case where the query is speech, a path can jump at most to a query phoneme which is three phonemes further along the sequence of query phonemes. Therefore, in this embodiment, mxhops has a value of four and the variable mx is set equal to four, provided there are four or more phonemes in the query, otherwise mx is set equal to the number of phonemes in the query. The processing then proceeds to steps s123, s125, s127, s129 and s131 which are operable to begin dynamic programming paths at each of the lattice points in the first four rows of the

search space by adding the corresponding transition probability to the score associated with the corresponding lattice point. When the query is generated from a spoken input, this ends the processing in step s103 shown in Figure 12 and the processing then proceeds to step s105.

In this embodiment, the system propagates the dynamic programming paths from the start points to the end points in step s105 by processing the lattice points in the search space column by column in a raster like technique. The control algorithm used to control this raster processing operation is shown in Figure 14. In step s151, the system compares an annotation phoneme loop pointer  $i$  with the number of phonemes in the annotation (Nann). Initially the annotation phoneme loop pointer  $i$  is set to zero and the processing will therefore initially proceed to step s153 where a similar comparison is made for a query phoneme loop pointer  $j$  relative to the total number of phonemes in the query (Nquery). Initially the loop pointer  $j$  is also set to zero and therefore the processing proceeds to step s155 where the system propagates the path ending at point  $(i,j)$  using the dynamic programming constraints discussed above. The way in which the system propagates the paths in step s155 will be described in more detail later. After step s155, the loop pointer  $j$  is incremented by one in step s157 and the processing returns to step s153. Once this processing has looped through all the phonemes in the query (thereby processing the current column of lattice points), the processing proceeds to step s159 where the query phoneme loop pointer  $j$  is reset to zero and the annotation phoneme loop pointer  $i$  is incremented by one.

The processing then returns to step s151 where a similar procedure is performed for the next column of lattice points. Once the last column of lattice points have been processed, the processing proceeds to step s161 where the annotation phoneme loop pointer  $i$  is reset to zero and the processing in step s105 shown in Figure 12 ends.

Figure 15 shows in more detail the processing steps involved in step s107 shown in Figure 12, when propagating the paths at the end points to the end null node  $\emptyset_e$ . As with the propagation from the start null node  $\emptyset_s$ , the lattice points which are the "end points", are defined by the dynamic programming constraints, which depend upon whether the query is text or speech. Further, in this embodiment, the dynamic programming constraints allow dynamic programming paths to exit the annotation at any point along the sequence of annotation phonemes. Therefore, if the query is text, then the system will allow dynamic programming paths ending in the last row of the lattice points, i.e. at points  $(i, N_{\text{query}}-1)$  for  $i = 0$  to  $N_{\text{ann}}-1$ , to propagate to the end null node  $\emptyset_e$ . If, however, the query was generated from speech, then the system allows any path propagating in the last four rows of the lattice points, i.e. points  $(i, N_{\text{query}}-4)$ ,  $(i, N_{\text{query}}-3)$ ,  $(i, N_{\text{query}}-2)$  and  $(i, N_{\text{query}}-1)$  for  $i = 0$  to  $N_{\text{ann}}-1$ , to propagate to the end null node  $\emptyset_e$ .

As shown in Figure 15, this process begins at step s171 where the system determines whether or not the query is text. If it is, then the processing proceeds to step s173 where the query phoneme loop pointer  $j$  is set to  $N_{\text{query}}-1$ . The processing then proceeds to step s175 where the annotation phoneme loop pointer  $i$  is compared

005507-102500

with the number of phonemes in the annotation (Nann). Initially the annotation phoneme loop pointer  $i$  is set to zero and therefore the processing will proceed to step s177 where the system calculates the transition score from point  $(i, Nquery-1)$  to the null end node  $\emptyset_e$ . This transition score is then combined with the cumulative score for the path ending at point  $(i, Nquery-1)$  which is stored in  $SCORE(i, Nquery-1)$ . As mentioned above, in this embodiment, the transition and cumulative scores are probability based and they are combined by multiplying the probabilities together. However, in this embodiment, in order to remove the need to perform multiplications and in order to avoid the use of high floating point precision, the system employs  $\log$  probabilities for the transition and cumulative scores. Therefore, in step s179, the system adds the cumulative score for the path ending at point  $(i, Nquery-1)$  to the transition score calculated in step s177 and the result is copied to a temporary store,  $TEMPENDSCORE$ .

As mentioned above, if two or more dynamic programming paths meet at the same point, then the cumulative scores for each of the paths are added together. Therefore, since  $\log$  probabilities are being used, the scores associated with paths which meet are effectively converted back to probabilities, added and then reconverted to  $\log$  probabilities. In this embodiment, this operation is referred to as a "log addition" operation. This is a well known technique and is described in, for example, the book entitled "Automatic Speech Recognition. The Development of the (Sphinx) System" by Lee, Kai-Fu published by Kluwer Academic Publishers, 1989, at pages 28 and 29.

Since the path propagating from point (i,Nquery-1) to the null end node will meet with other dynamic programming paths, the system performs a log addition of TEMPENDSCORE with the score stored in the end node (ENDSCORE) and the result is stored in ENDSCORE. The processing then proceeds to step s183 where the annotation phoneme loop pointer i is incremented. The processing then returns to step s175 where a similar process is performed for the next lattice point in the last row of lattice points. Once all the lattice points in the last row have been processed in this way, the processing performed in step s107 shown in Figure 12 ends.

If the system determines at step s171 that the query is not text, then the processing proceeds to step s185 where the query phoneme loop pointer j is set to the number of phonemes in the query minus mxhops, i.e. Nquery-4. The processing then proceeds to step s187, where the system checks to see if the annotation phoneme loop pointer i is less than the number of phonemes in the annotation (Nann). Initially the annotation phoneme loop pointer i is set to zero and therefore the processing proceeds to step s189 where the system checks to see if the query phoneme loop pointer j is less than the number of phonemes in the query (Nquery). Initially it will be, and the processing proceeds to step s191 where the system calculates the transition score from lattice point (i,j) to the null end node  $\emptyset_0$ . This transition score is then added, in step s193, to the cumulative score for the path ending at point (i,j) and the result is copied to the temporary score, TEMPENDSCORE. The processing then proceeds to step s195 where the system performs a log addition of TEMPENDSCORE with ENDSCORE and the result is

005507 105500

stored in ENDSCORE. The processing then proceeds to step s197 where the query phoneme loop pointer  $j$  is incremented by one and the processing returns to step s189. The above processing steps are then repeated until the query phoneme loop pointer  $j$  has been incremented so that it equals the number of phonemes in the query ( $N_{\text{query}}$ ). The processing then proceeds to step s199, where the query phoneme loop pointer  $j$  is reset to  $N_{\text{query}}-4$  and the annotation phoneme loop pointer  $i$  is incremented by one. The processing then returns to step s187. The above processing steps are then repeated until all the lattice points in the last four rows of the search space have been processed in this way, after which the processing performed in step s107 shown in Figure 12 ends.

### *Propagate*

In step s155 shown in Figure 14, the system propagates the path ending at lattice point  $(i,j)$  using the dynamic programming constraints discussed above. Figure 16 is a flowchart which illustrates the processing steps involved in performing this propagation step. As shown, in step s211, the system sets the values of two variables  $mx_i$  and  $mx_j$  and initialises annotation phoneme loop pointer  $i_2$  and query phoneme loop pointer  $j_2$ . The loop pointers  $i_2$  and  $j_2$  are provided to loop through all the lattice points to which the path ending at point  $(i,j)$  can propagate to and the variables  $mx_i$  and  $mx_j$  are used to ensure that  $i_2$  and  $j_2$  can only take the values which are allowed by the dynamic programming constraints. In particular,  $mx_i$  is set equal to  $i$  plus  $mx_{\text{hops}}$ , provided this is less than or equal to the number of phonemes in the annotation, otherwise  $mx_i$  is set equal to the number

005507 44056960

of phonemes in the annotation (Nann). Similarly,  $mxj$  is set equal to  $j$  plus  $mxhops$ , provided this is less than or equal to the number of phonemes in the query, otherwise  $mxj$  is set equal to the number of phonemes in the query (Nquery). Finally, in step s211, the system initialises the annotation phoneme loop pointer  $i2$  to be equal to the current value of the annotation phoneme loop pointer  $i$  and the query phoneme loop pointer  $j2$  to be equal to the current value of the query phoneme loop pointer  $j$ .

Since the dynamic programming constraints employed by the system depend upon whether the annotation is text or speech and whether the query is text or speech, the next step is to determine how the annotation and the query were generated. This is performed by the decision blocks s213, s215 and s217. If both the annotation and the query are generated from speech, then the dynamic programming path ending at lattice point  $(i,j)$  can propagate to the other points shown in Figure 11 and process steps s219 to s235 operate to propagate this path to these other points. In particular, in step s219, the system compares the annotation phoneme loop pointer  $i2$  with the variable  $mx_i$ . Since annotation phoneme loop pointer  $i2$  is set to  $i$  and  $mx_i$  is set equal to  $i+4$ , in step s211, the processing will proceed to step s221 where a similar comparison is made for the query phoneme loop pointer  $j2$ . The processing then proceeds to step s223 which ensures that the path does not stay at the same lattice point  $(i,j)$  since initially,  $i2$  will equal  $i$  and  $j2$  will equal  $j$ . Therefore, the processing will initially proceed to step s225 where the query phoneme loop pointer  $j2$  is incremented by one.

005507 10500

The processing then returns to step s221 where the incremented value of  $j2$  is compared with  $mxj$ . If  $j2$  is less than  $mxj$ , then the processing returns to step s223 and the processing proceeds to step s227, which is operable to prevent too large a hop along both the sequences of annotation phonemes and query phonemes. It does this by ensuring that the path is only propagated if  $i2 + j2$  is less than  $i + j + mxhops$ . This ensures that only the triangular set of points shown in Figure 11 are processed. Provided this condition is met, the processing then proceeds to step s229 where the system calculates the transition score (TRANSORE) from lattice point  $(i,j)$  to lattice point  $(i2,j2)$ . The processing then proceeds to step s231 where the system adds the transition score determined in step s229 to the cumulative score stored for the point  $(i,j)$  and copies this to a temporary store, TEMPSCORE. As mentioned above, in this embodiment, if two or more dynamic programming paths meet at the same lattice point, the cumulative scores associated with each of the paths are added together. Therefore, in step s233, the system performs a log addition of TEMPSCORE with the cumulative score already stored for point  $(i2,j2)$  and the result is stored in SCORE  $(i2,j2)$ . The processing then returns to step s225 where the query phoneme loop pointer  $j2$  is incremented by one and the processing returns to step s221. Once the query phoneme loop pointer  $j2$  has reached the value of  $mxj$ , the processing proceeds to step s235, where the query phoneme loop pointer  $j2$  is reset to the initial value  $j$  and the annotation phoneme loop pointer  $i2$  is incremented by one. The processing then proceeds to step s219 where the processing begins again for the next column of points shown in Figure 11. Once the path



has been propagated from point (i,j) to all the other points shown in Figure 11, the processing ends.

If the decision blocks s213 and s215 determine that the annotation is text and the query is speech, then the processing proceeds to steps s241 to s251, which are operable to propagate the path ending at point (i,j) to the points shown in Figure 9a. In particular, in step s241, the system determines whether or not the annotation phoneme loop pointer i is pointing to the last phoneme in the annotation. If it is, then there are no more phonemes in the annotation and the processing ends. If the annotation phoneme loop pointer i is less than Nann-1, then the processing proceeds to step s243, where the query phoneme loop pointer j2 is compared with mxj. Initially, j2 will be less than mxj and therefore the processing proceeds to step s245 where the system calculates the transition score (TRANSCORE) from point (i,j) to point (i+1,j2). This transition score is then added to the cumulative score associated with the path ending at point (i,j) and the result is copied to the temporary score, TEMPSCORE. The system then performs, in step s249, a log addition of TEMPSCORE with the cumulative score associated with the point (i+1,j2) and stores the result in SCORE (i+1,j2), to ensure that the path scores for paths which meet at the lattice point (i+1,j2) are combined. The processing then proceeds to step s251 where the query phoneme loop pointer j2 is incremented by one and then the processing returns to step s243. Once the path which ends at point (i,j) has been propagated to the other points shown in Figure 9a, j2 will equal mxj and the propagation of the path ending at point (i,j) will end.

005507 105500

If the decision blocks s213 and s217 determine that the annotation is speech and the query is text, then the processing proceeds to steps s255 to s265 shown in Figure 16b, which are operable to propagate the path ending at point (i,j) to the other points shown in Figure 9b. This is achieved by firstly checking, in step s255, that the query phoneme loop pointer j is not pointing to the last phoneme in the sequence of phonemes representing the query. If it is not, then the processing proceeds to step s257 where the annotation phoneme loop pointer i2 is compared with mxi. Initially i2 has a value of i and provided annotation phoneme i is not at the end of the sequence of phonemes representing the annotation, the processing will proceed to step s259, where the transition score for moving from point (i,j) to point (i2,j+1) is calculated. The processing then proceeds to step s261 where this transition score is added to the cumulative score for the path ending at point (i,j) and the result is copied to the temporary score, TEMPSCORE. The processing then proceeds to step s263 where a log addition is performed of TEMPSCORE with the cumulative score already stored for the point (i2,j+1) and the result is stored in SCORE (i2,j+1). The processing then proceeds to step s265 where the annotation phoneme loop pointer i2 is incremented by one and the processing returns to step s257. These processing steps are then repeated until the path ending at point (i,j) has been propagated to each of the other points shown in Figure 9b. At this time, the propagation of the path at point (i,j) is completed and the processing ends.

Finally, if the decision blocks s213 and s215 determine that both the annotation and the query are text, then the

005507-10500

processing proceeds to steps s271 to s279 shown in Figure 16b, which are operable to propagate the path ending at point (i,j) to the point (i+1,j+1), provided of course there is a further annotation phoneme and a further query phoneme. In particular, in step s271, the system checks that the annotation phoneme loop pointer i is not pointing to the last annotation phoneme. If it is not then the processing proceeds to step s273 where a similar check is made for the query phoneme loop pointer j relative to the sequence of query phonemes. If there are no more annotation phonemes or if there are no more query phonemes, then the processing ends. If, however, there is a further annotation phoneme and a further query phoneme, then the processing proceeds to step s275 where the system calculates the transition score from point (i,j) to point (i+1,j+1). This transition score is then added, in step s277, to the cumulative score stored for the point (i,j) and stored in the temporary score, TEMPSCORE. The processing then proceeds to step s279 where the system performs a log addition of TEMPSCORE with the cumulative score already stored for point (i+1,j+1) and the result is copied to SCORE (i+1,j+1). As those skilled in the art will appreciate, steps s277 and s279 are necessary in this embodiment because the dynamic programming constraints allow a path to start at any phoneme within the sequence of phonemes representative of the annotation and therefore, point (i+1,j+1) may already have a score associated with it. After step s279, the propagation of point (i,j) is completed and the processing ends.

#### *Transition Score*

In steps s103, s105 and s107 shown in Figure 12, dynamic

005507-10500

programming paths are propagated and during this propagation, the transition score from one point to another point is calculated in steps s127, s117, s177, s191, s229, s245, s259 and s275. In these steps, the system calculates the appropriate insertion probabilities, deletion probabilities and decoding probabilities relative to the start point and end point of the transition. The way in which this is achieved in this embodiment, will now be described with reference to Figures 17 and 18.

In particular, Figure 17 shows a flow diagram which illustrates the general processing steps involved in calculating the transition score for a path propagating from lattice point (i,j) to lattice point (i2,j2). In step s291, the system calculates, for each annotation phoneme which is inserted between point (i,j) and point (i2,j2), the score for inserting the inserted phoneme(s) (which is just the log of probability  $PI( )$  discussed above) and adds this to an appropriate store, INSERTSCORE. The processing then proceeds to step s293 where the system performs a similar calculation for each query phoneme which is inserted between point (i,j) and point (i2,j2) and adds this to INSERTSCORE. Note, however, that if (i,j) is the start null node  $\emptyset_s$  or if (i2,j2) is the end null node  $\emptyset_e$ , then the system does not calculate the insertion probabilities for any inserted annotation phonemes (since there is no penalty for starting or ending a path at any of the annotation phonemes), although it does calculate insertion probabilities for any inserted query phonemes. As mentioned above, the scores which are calculated are log based probabilities, therefore the addition of the scores

005507-10500

in INSERTSCORE corresponds to the multiplication of the corresponding insertion probabilities. The processing then proceeds to step s295 where the system calculates the scores for any deletions and/or any decodings in propagating from point (i,j) to point (i2,j2) and these scores are added and stored in an appropriate store, DELSCORE. The processing then proceeds to step s297 where the system adds INSERTSCORE and DELSCORE and copies the result to TRANSCORE.

The processing involved in step s295 to determine the deletion and/or decoding scores in propagating from point (i,j) to point (i2,j2) will now be described in more detail with reference to Figure 18. Since the possible deletions and decodings depend on whether or not the annotation was generated from text and whether or not the query was generated from text, the decision blocks s301, s303 and s305 determine if the annotation is text or speech and if the query is text or speech. If these decision blocks determine that both the annotation and the query are text, then there are no deletions and the decoding of the two phonemes is performed by a boolean match in step s307. If annotation phoneme  $a_{i2}$  is the same as query phoneme  $q_{j2}$ , then the processing proceeds to step s309, where TRANSCORE is set to equal log [one] (i.e. zero) and the processing ends. If, however, annotation phoneme  $a_{i2}$  is not the same as query phoneme  $q_{j2}$ , then the processing proceeds to step s311 where TRANSCORE is set to a very large negative number which is a system approximation of log [zero] and the processing then ends.

If the decision blocks s301 and s305 determine that the

00507 10500

annotation is speech and the query is text, then the transition scores are determined using the simplified form of equation (4) discussed above. In this case, the processing passes from step s303 to step s313 where the system determines if annotation loop pointer i2 equals annotation loop pointer i. If it does, then this means that the path has propagated from point (i,j) to point (i,j+1). Therefore, the query phoneme  $q_{j+1}$  has been deleted from the sequence of annotation phonemes relative to the sequence of query phonemes. Therefore, in step s317, the system copies the log probability of deleting phoneme  $q_{j+1}$  (i.e.  $\log P(\emptyset | q_{j+1}, C)$ ) to DELSCORE and the processing ends. If in step s313, the system determines that i2 is not equal to i, then the system is considering the propagation of the path ending at point (i,j) to one of the points (i+1,j+1), (i+2,j+1) or (i+3,j+1). In which case, there are no deletions, only insertions and a decoding between annotation phoneme  $a_{i2}$  with query phoneme  $q_{j+1}$ . Therefore, in step s315, the system copies the log probability of decoding query phoneme  $q_{j+1}$  as annotation phoneme  $a_{i2}$  (i.e.  $\log P(a_{i2} | q_{j+1}, C)$ ) to DELSCORE and the processing ends.

If the decision blocks s301 and s305 determine that the annotation is text and that the query is speech, then the transition scores are determined using the other simplified form of equation (4) discussed above. In this case, the processing passes from step s305 to step s319 where the system determines whether or not query phoneme loop pointer j2 equals query phoneme loop pointer j. If it does, then the system is calculating the transition score from point (i,j) to point (i+1,j). In this case, the annotation phoneme  $a_{i+1}$  has been deleted from the

00507 102500

sequence of query phonemes relative to the sequence of annotation phonemes. Therefore, in step s321, the system determines and copies the log probability of deleting annotation phoneme  $a_{i+1}$  (i.e.  $\log P(\emptyset|a_{i+1},C)$ ) to DELSCORE and then the processing ends. If at step s319, the system determines that query phoneme loop pointer  $j_2$  is not equal to query phoneme loop pointer  $j$ , then the system is currently determining the transition score from point  $(i,j)$  to one of the points  $(i+1,j+1)$ ,  $(i+1,j+2)$  or  $(i+1,j+3)$ . In this case, there are no deletions, only insertions and a decoding between annotation phoneme  $a_{i+1}$  with query phoneme  $q_{j_2}$ . Therefore, in step s323, the system determines and copies the log probability of decoding annotation phoneme  $a_{i+1}$  as query phoneme  $q_{j_2}$  (i.e.  $\log P(q_{j_2}|a_{i+1},C)$ ) to DELSCORE and the processing ends.

If the decision blocks s301 and s303 determine that both the annotation and the query are generated from speech, then the transition scores are determined using equation (4) above. In this case, the processing passes from step s303 to step s325 where the system determines if the annotation loop pointer  $i_2$  equals annotation loop pointer  $i$ . If it does, then the processing proceeds to step s327 where a phoneme loop pointer  $r$  is initialised to one. The phoneme pointer  $r$  is used to loop through each possible phoneme known to the system during the calculation of equation (4) above. The processing then proceeds to step s329, where the system compares the phoneme pointer  $r$  with the number of phonemes known to the system,  $N_{\text{phonemes}}$  (which in this embodiment equals 43). Initially  $r$  is set to one in step s327, therefore the processing proceeds to step s331 where the system determines the log probability of phoneme  $p_r$  occurring

005207 42056960

(i.e.  $\log P(p_r|C)$ ) and copies this to a temporary score TEMPDELScore. If annotation phoneme loop pointer i2 equals annotation phoneme i, then the system is propagating the path ending at point (i,j) to one of the points (i,j+1), (i,j+2) or (i,j+3). Therefore, there is a phoneme in the query which is not in the annotation. Consequently, in step s333, the system adds the log probability of deleting phoneme  $p_r$  from the annotation (i.e.  $\log P(\emptyset|p_r,C)$ ) to TEMPDELScore. The processing then proceeds to step s335, where the system adds the log probability of decoding phoneme  $p_r$  as query phoneme  $q_{j2}$  (i.e.  $\log P(q_{j2}|p_r,C)$ ) to TEMPDELScore. The processing then proceeds to step s337 where the log addition of TEMPDELScore and DELScore is performed and the result is stored in DELScore. The processing then proceeds to step s339 where the phoneme loop pointer r is incremented by one and then the processing returns to step s329 where a similar processing is performed for the next phoneme known to the system. Once this calculation has been performed for each of the 43 phonemes known to the system, the processing ends.

If at step s325, the system determines that i2 is not equal to i, then the processing proceeds to step s341 where the system determines if the query phoneme loop pointer j2 equals query phoneme loop pointer j. If it does, then the processing proceeds to step s343 where the phoneme loop pointer r is initialised to one. The processing then proceeds to step s345 where the phoneme loop pointer r is compared with the total number of phonemes known to the system (Nphonemes). Initially r is set to one in step s343, and therefore, the processing proceeds to step s347 where the log probability of



phoneme  $p_r$  occurring is determined and copied into the temporary store TEMPDELScore. The processing then proceeds to step s349 where the system determines the log probability of decoding phoneme  $p_r$  as annotation phoneme  $a_{i2}$  and adds this to TEMPDELScore. If the query phoneme loop pointer  $j2$  equals query phoneme loop pointer  $j$ , then the system is propagating the path ending at point  $(i,j)$  to one of the points  $(i+1,j)$ ,  $(i+2,j)$  or  $(i+3,j)$ . Therefore, there is a phoneme in the annotation which is not in the query. Consequently, in step s351, the system determines the log probability of deleting phoneme  $p_r$  from the query and adds this to TEMPDELScore. The processing then proceeds to step s353 where the system performs the log addition of TEMPDELScore with DELScore and stores the result in DELScore. The phoneme loop pointer  $r$  is then incremented by one in step s355 and the processing returns to step s345. Once the processing steps s347 to s353 have been performed for all the phonemes known to the system, the processing ends.

If at step s341, the system determines that query phoneme loop pointer  $j2$  is not equal to query phoneme loop pointer  $j$ , then the processing proceeds to step s357 where the phoneme loop pointer  $r$  is initialised to one. The processing then proceeds to step s359 where the system compares the phoneme counter  $r$  with the number of phonemes known to the system ( $N_{\text{phonemes}}$ ). Initially  $r$  is set to one in step s357, and therefore, the processing proceeds to step s361 where the system determines the log probability of phoneme  $p_r$  occurring and copies this to the temporary score TEMPDELScore. If the query phoneme loop pointer  $j2$  is not equal to query phoneme loop pointer  $j$ , then the system is propagating the path ending

005207 24056960

at point  $(i,j)$  to one of the points  $(i+1,j+1)$ ,  $(i+1,j+2)$  and  $(i+2,j+1)$ . Therefore, there are no deletions, only insertions and decodings. The processing therefore proceeds to step s363 where the log probability of decoding phoneme  $p_r$  as annotation phoneme  $a_{i2}$  is added to TEMPDELScore. The processing then proceeds to step s365 where the log probability of decoding phoneme  $p_r$  as query phoneme  $q_{j2}$  is determined and added to TEMPDELScore. The system then performs, in step s367, the log addition of TEMPDELScore with DELScore and stores the result in DELScore. The phoneme counter  $r$  is then incremented by one in step s369 and the processing returns to step s359. Once processing steps s361 to s367 have been performed for all the phonemes known to the system, the processing ends.

#### NORMALISATION

The above description of the dynamic programming process has dealt only with the numerator part of equation (3) above. Therefore, after an input query has been matched with a sequence of annotation phonemes in the database, the score for that match (which is stored in ENDScore) must be normalised by the normalising term defined by the denominator of equation (3). As mentioned above, the calculation of the denominator term is performed at the same time as the calculation of the numerator, i.e. in the dynamic programming routine described above. This is because, as can be seen from a comparison of the numerator and the denominator, the terms required for the denominator are all calculated on the numerator. It should, however, be noted that when the annotation or the query is generated from text, no normalisation is performed. In this embodiment, normalisation is

005507 2056960

performed so that longer annotations are not given more weight than shorter annotations and so that annotations which include common phonemes are not given more weight than annotations which include uncommon phonemes. It does this by normalising the score by a term which depends upon how well the annotation matches the underlying model.

### TRAINING

In the above embodiment, the system used 1892 decoding/deletion probabilities and 43 insertion probabilities (referred to above as the confusion statistics) which were used to score the dynamic programming paths in the phoneme matching operation. In this embodiment, these probabilities are determined in advance during a training session and are stored in a memory (not shown). In particular, during this training session, a speech recognition system is used to provide a phoneme decoding of speech in two ways. In the first way, the speech recognition system is provided with both the speech and the actual words which are spoken. The speech recognition unit can therefore use this information to generate the canonical phoneme sequence of the spoken words to obtain an ideal decoding of the speech. The speech recognition system is then used to decode the same speech, but this time without knowledge of the actual words spoken (referred to hereinafter as the free decoding). The phoneme sequence generated from the free decoding will differ from the canonical phoneme sequence in the following ways:

- i) the free decoding may make mistakes and insert phonemes into the decoding which are not present in

005207 405090

the canonical sequence or, alternatively, omit phonemes in the decoding which are present in the canonical sequence;

- ii) one phoneme may be confused with another; and
  - 5 iii) even if the speech recognition system decodes the speech perfectly, the free decoding may nonetheless differ from the canonical decoding due to the differences between conversational pronunciation and canonical pronunciation, e.g., in
- 10 conversational speech the word "and" (whose canonical forms are /ae/ /n/ /d/ and /ax/ /n/ /d/) is frequently reduced to /ax/ /n/ or even /n/.

Therefore, if a large number of utterances are decoded

15 into their canonical forms and their free decoded forms, then a dynamic programming method can be used to align the two. This provides counts of what was decoded, d, when the phoneme should, canonically, have been a p. From these training results, the above decoding, deletion

20 and insertion probabilities can be approximated in the following way.

The probability that phoneme, d, is an insertion is given by:

$$P(d|C) = \frac{I_d}{n_o^d} \quad (9)$$

where  $I_d$  is the number of times the automatic speech recognition system inserted phoneme d and  $n_o^d$  is the total number of decoded phonemes which are inserted relative to the canonical sequence.

005507 102500

The probability of decoding phoneme  $p$  as phoneme  $d$  is given by:

$$P(d|p,C) = \frac{c_{dp}}{n_p} \quad (10)$$

where  $c_{dp}$  is the number of times the automatic speech recognition system decoded  $d$  when it should have been  $p$  and  $n_p$  is the number of times the automatic speech recognition system decoded anything (including a deletion) when it should have been  $p$ .

The probability of not decoding anything (i.e. there being a deletion) when the phoneme  $p$  should have been decoded is given by:

$$P(\emptyset|p,C) = \frac{O_p}{n_p} \quad (11)$$

where  $O_p$  is the number of times the automatic speech recognition system decoded nothing when it should have decoded  $p$  and  $n_p$  is the same as above.

#### PHONEME CONFIDENCE

In the above embodiment, a speech query was input to a speech recognition system which generated a sequence of phonemes representative of the spoken query. As those skilled in the art of speech recognition systems will be aware, many phoneme speech recognition systems output a best guess phoneme together with some alternatives. For example, a speech recognition system might output that

the current portion of the input speech corresponds to phoneme /eh/ with confidence probability 0.4 but that it could have been phoneme /e/ with confidence probability 0.3 or phoneme /ey/ with confidence probability 0.3.

5

If this information is to be incorporated into the dynamic programming matching algorithm described above, then equation (4) above has to be modified. This could be done by summing the probabilities given in equation (4) for each of the phoneme possibilities (i.e. adding a further summation to the equation). However, summing over phoneme confidence as well as confusion information would slow the algorithm down by an order of magnitude. An alternative method, and the one used in this embodiment, is to combine the confidence and confusion information into a single calculation. This is possible, because they both represent estimates of the same thing - the underlying probability of a particular phoneme being decoded (by the speech recognition system) as any other phoneme. In this method, the system aligns the sequence of best guess query phonemes with the sequence of best guess annotation phonemes and uses the confidence probabilities generated for each phoneme to modify the confidence probabilities calculated at each lattice point of the dynamic programming alignment algorithm.

10

15

20

25

Therefore, introducing the confidence information into equation (4), the term calculated at lattice point  $(a_i, q_j)$  is:

30

$$\sum_{r=1}^{N_p} P(a_i|p_r, C, e_{ai})P(q_j|p_r, C, e_{qj})P(p_r|C) \quad (12)$$

00520T 44056960

on the numerator, and

$$\sum_{r=1}^{N_p} P(a_i|p_r, C, e_{ai}) P(p_r|C) \quad (13)$$

on the denominator, where  $e_{ai}$  is the confidence information for the best guess annotation phoneme  $a_i$  and  $e_{qj}$  is the confidence information for the best guess query phoneme  $q_j$ . Each of  $e_{ai}$  and  $e_{qj}$  will comprise a vector of  $N_p$  confidence values - one for each of the phonemes known to the system, although most of the entries in these vectors will be zero as the recognition system will only, usually, output a few alternatives to the best guess phoneme. In this embodiment, the following equation is used to determine  $P(q_j|p_r, C, e_{qj})$ :

$$P(q_j|p_r, C, e_{qj}) = \frac{b_c c_{jr} + b_e e_{qj}^r + \beta}{b_c n_r + b_e + N_p \beta} \quad (14)$$

where  $c_{jr}$  and  $n_r$  are the counts generated during the training routine discussed above for the number of times the automatic speech recognition system decoded phoneme  $q_j$  when it should have decoded phoneme  $p_r$  and the number of times the automatic speech recognition system decoded anything (including a deletion) when it should have been phoneme  $p_r$ , respectively;  $e_{qj}^r$  is the confidence probability for the current model phoneme ( $p_r$ ) being evaluated which was output by the speech recognition system when it decoded the current query phoneme  $q_j$ ;  $\beta$  represents a lower limit of the confidence probabilities and depends upon any pruning threshold which is used in the phoneme decoder of the speech recognition system;  $N_p$  is the total number of phonemes in the system (which in this embodiment is 43); and  $b_c$  and  $b_e$  are scaling factors which are applied to the confusion counts and confidence

005207 4056960

values respectively. From the relative size of the confidence probabilities and confusion probabilities, we know that  $b_e$  should be of the order of several hundred and  $b_c$  should be of the order of several hundredths. Experiments reveal that values of  $b_e = 700$ ,  $b_c = 0.01$  and  $\beta = 0.001$  or less to be appropriate.

A similar expression is calculated for  $P(a_i|p_r, C, e_{ai})$  except using the confidence information ( $e_{ai}$ ) output by the speech recognition system during the decoding of the current annotation phoneme  $a_i$ . The rest of the dynamic programming alignment algorithm is the same as in the first embodiment described above and will not therefore be described again.

#### ALTERNATIVE EMBODIMENTS

In the above embodiment, the confusion and confidence probabilities were combined using equation (14) above. In an alternative embodiment, a linear combination, such as:

$$P(q_i|p_r, C, e_{qi}) = \lambda P(q_i|p_r, C) + (1 - \lambda)e_{qi}^r \quad (15)$$

could be used, where  $\lambda$  is a constant. Alternatively still, a non-linear combination of the two probabilities could be used. However, the combination performed using equation (14) is preferred, since it is effectively a weighted linear combination of the two probabilities, with the weighting applied to the confidence probability ( $e$ ) being dependent upon the confusion statistics. In particular, the weighting applied to the confidence probability varies as  $1/n_r$ . As a result, when the confusion statistics are unreliable, because of a lack of

005207 440956960



training data, more weight is given to the confidence probabilities. Whereas, when there is a lot of training data for the current model phoneme ( $p_r$ ), more reliance is placed on the confusion statistics than the confidence probabilities.

In the above embodiment, confidence probabilities for both the query and the annotation were used to modify the score which is calculated during the dynamic programming alignment routine. As those skilled in the art will appreciate, if this confidence information is only available for one of the input query or the annotation, then only the relevant confusion probabilities would be amended.

As those skilled in the art will appreciate, the above technique for matching one sequence of phonemes with another sequence of phonemes can be applied to applications other than data retrieval. Additionally, as those skilled in the art will appreciate, although the system described above has used phonemes in the phoneme and word lattice, other phoneme-like units can be used, such as syllables or katakana (Japanese alphabet).

As those skilled in the art will appreciate, the above description of the dynamic programming matching and alignment of the two sequences of phonemes was given by way of example only and various modifications can be made. For example, whilst a raster scanning technique for propagating the paths through the lattice points was employed, other techniques could be employed which progressively propagates the paths through the lattice points. Additionally, as those skilled in the art will

appreciate, dynamic programming constraints other than those described above may be used to control the matching process.

5 In the above embodiment, the annotation was generally longer than the query and the dynamic programming alignment algorithm aligned the query with the entire annotation. In an alternative embodiment, the alignment  
10 algorithm may compare the query with the annotation by stepping the query over the annotation from beginning to end and, at each step, comparing the query with a portion of the annotation of approximately the same size as the query. In such an embodiment, at each step, the query would be aligned with the corresponding portion of the  
15 annotation using a similar dynamic programming technique to the one described above. This technique is illustrated in Figure 19a with the resulting plot showing the way in which the dynamic programming score for the alignments between the query and a current annotation varies as the query is stepped over the annotation shown  
20 in Figure 19b. The peaks in the plot shown in Figure 19b represent the parts of the annotation which match best with the query. The annotation which is most similar to the query can then be determined by comparing the peak DP score obtained during the comparison of the query with  
25 each annotation.

In the above embodiment, pictures were annotated using the phoneme and word lattice annotation data. As those  
30 skilled in the art will appreciate, this phoneme and word lattice data can be used to annotate many different types of data files. For example, this kind of annotation data can be used in medical applications for annotating x-rays

005207 102500

of patients, 3D videos of, for example, NMR scans, ultrasound scans etc. It can also be used to annotate 1D data, such as audio data or seismic data.

5 In the above embodiments, a speech recognition system which generates a sequence of phonemes from the input speech signal was used. As those skilled in the art will appreciate, the above system can be used with other types of speech recognition systems which generate, for  
10 example, a sequence of output words or a word lattice which can then be decomposed into a corresponding string of phonemes with alternatives, in order to simulate a recogniser which produces phoneme strings.

15 In the above embodiment, the insertion, deletion and decoding probabilities were calculated from the confusion statistics for the speech recognition system using a maximum likelihood estimate of the probabilities. As those skilled in the art of statistics will appreciate,  
20 other techniques, such as maximum entropy techniques, can be used to estimate these probabilities. Details of a suitable maximum entropy technique can be found at pages 45 to 52 in the book entitled "Maximum Entropy and Bayesian Methods" published by Kluwer Academic publishers  
25 and written by John Skilling, the contents of which are incorporated herein by reference.

In the above embodiment, the database 29 and the automatic speech recognition unit 51 were both located  
30 within the user terminal 59. As those skilled in the art will appreciate, this is not essential. Figure 20 illustrates an embodiment in which the database 29 and the search engine 53 are located in a remote server 60

005507-10500

and in which the user terminal 59 accesses the database 29 via the network interface units 67 and 69 and a data network 68 (such as the Internet). In this embodiment, the user terminal 59 can only receive voice queries from the microphone 7. These queries are converted into phoneme and word data by the automatic speech recognition unit 51. This data is then passed to the control unit 55 which controls the transmission of the data over the data network 68 to the search engine 53 located within the remote server 60. The search engine 53 then carries out the search in a similar manner to the way in which the search was performed in the above embodiment. The results of the search are then transmitted back from the search engine 53 to the control unit 55 via the data network 68. The control unit 55 then considers the search results received back from the network and displays appropriate data on the display 57 for viewing by the user 39.

In addition to locating the database 29 and the search engine 53 in the remote server 60, it is also possible to locate the automatic speech recognition unit 51 in the remote server 60. Such an embodiment is shown in Figure 21. As shown, in this embodiment, the input voice query from the user is passed via input line 61 to a speech encoding unit 73 which is operable to encode the speech for efficient transfer through the data network 68. The encoded data is then passed to the control unit 55 which transmits the data over the network 68 to the remote server 60, where it is processed by the automatic speech recognition unit 51. The phoneme and word data generated by the speech recognition unit 51 for the input query is then passed to the search engine 53 for use in searching

the database 29. The search results generated by the search engine 53 are then passed, via the network interface 69 and the network 68, back to the user terminal 59. The search results received back from the remote server are then passed via the network interface unit 67 to the control unit 55 which analyses the results and generates and displays appropriate data on the display 57 for viewing by the user 39.

In a similar manner, a user terminal 59 may be provided which only allows typed inputs from the user and which has the search engine and the database located in the remote server. In such an embodiment, the phonetic transcription unit 75 may be located in the remote server 60 as well.

In the above embodiments, a dynamic programming algorithm was used to align the sequence of query phonemes with the sequences of annotation phonemes. As those skilled in the art will appreciate, any alignment technique could be used. For example, a naive technique could be used which identifies all possible alignments. However, dynamic programming is preferred because of its ease of implementation using standard processing hardware.

A description has been given above of the way in which two or more canonical sequences of phonemes are compared using a dynamic programming technique. However, as shown in Figures 2 and 3, the annotations are preferably stored as lattices. As those skilled in the art will appreciate, in order that the above comparison techniques will work with these lattices, the phoneme sequences defined by the lattices must be "flattened" into a single

005207 405000

sequence of phonemes with no branches. A naive approach to do this would be to identify all the different possible phoneme sequences defined by the lattice and then to compare each of those with the or each query sequence. However, this is not preferred, since common parts of the lattice will be matched several times with each query sequence. Therefore, the lattice is preferably flattened by sequentially labelling each phoneme within the lattice in accordance with the time stamp information available for each phoneme within the lattice. Then, during the dynamic programming alignment, different dynamic programming constraints are used at each DP lattice point, in order to ensure that the paths propagate in accordance with the lattice structure.

The table below illustrates the DP constraints used for part of the phoneme lattice shown in Figure 2. In particular, the first column illustrates the phoneme number ( $p_1$  to  $p_9$ ) assigned to each phoneme in the lattice; the middle column corresponds to the actual phoneme in the lattice; and the last column illustrates, for each phoneme, the phonemes to which a path ending at that phoneme may propagate to, at the next dynamic programming time point. Although not shown, the middle column will also include details of the node to which the phoneme is connected and the corresponding phoneme link.

| Phoneme Number | Phoneme | Dynamic Programming Constraints |
|----------------|---------|---------------------------------|
| $p_1$          | /p/     | $p_1; p_2; p_3; p_4$            |
| $p_2$          | /ih/    | $p_2; p_3; p_4; p_5$            |

|                |      |   |
|----------------|------|---|
| p <sub>3</sub> | /k/  | p <sub>6</sub><br>p <sub>3</sub> ; p <sub>4</sub> ; p <sub>5</sub> ; p <sub>7</sub><br>p <sub>8</sub>   |
| p <sub>4</sub> | /ch/ | p <sub>6</sub> ; p <sub>10</sub><br>p <sub>4</sub> ; p <sub>5</sub> ; p <sub>7</sub> ; p <sub>9</sub><br>p <sub>8</sub> ; p <sub>11</sub>   |
| p <sub>5</sub> | /ax/ | p <sub>6</sub> ; p <sub>10</sub> ; p <sub>12</sub><br>p <sub>5</sub> ; p <sub>7</sub> ; p <sub>9</sub> ; p <sub>12</sub><br>p <sub>8</sub> ; p <sub>11</sub> ; p <sub>13</sub><br>p <sub>14</sub> |
| p <sub>6</sub> | /ax/ | p <sub>6</sub> ; p <sub>10</sub> ; p <sub>12</sub> ; p <sub>15</sub><br>p <sub>16</sub>   |
| p <sub>7</sub> | /ao/ | p <sub>7</sub> ; p <sub>9</sub> ; p <sub>12</sub> ; p <sub>15</sub><br>p <sub>16</sub>  |
| p <sub>8</sub> | /ah/ | p <sub>8</sub> ; p <sub>11</sub> ; p <sub>13</sub> ; p <sub>18</sub><br>p <sub>14</sub> ; p <sub>17</sub>   |
| p <sub>9</sub> | /f/  | p <sub>9</sub> ; p <sub>12</sub> ; p <sub>15</sub> ; p <sub>18</sub><br>p <sub>16</sub> ; p <sub>18</sub>   |

5

10

15

20

For example, if a dynamic programming path ends at time ordered phoneme p<sub>4</sub>, then that dynamic programming path can stay at phoneme p<sub>4</sub> or it can propagate to any of time ordered phonemes p<sub>5</sub> to p<sub>11</sub>. As shown in the table, at some points the possible phonemes to which a path can extend are not consecutively arranged in the time ordered phoneme sequence. For example, for a dynamic programming path ending at time ordered phoneme p<sub>6</sub>, this path can either stay at this phoneme or progress to phonemes p<sub>10</sub>, p<sub>12</sub>, p<sub>15</sub> or p<sub>16</sub>. By consecutively numbering the phonemes in the lattice in this way and by varying the dynamic programming constraints used in dependence upon the

lattice, an efficient dynamic programming matching between the input query and the annotation lattice can be achieved. Further, as those skilled in the art will appreciate, if the input query also generates a lattice,  
5 then this may be flattened in a similar way and the dynamic programming constraints adjusted accordingly.

In the above embodiment, the same phoneme confusion probabilities were used for both the annotations and the queries. As those skilled in the art will appreciate, if  
10 different recognition systems are used to generate these, then different phoneme confusion probabilities should be used for the annotations and the queries. Since these confusion probabilities depend upon the recognition  
15 system that was used to generate the phoneme sequences.

In the above embodiment, when either the annotation or the query was generated from text, it was assumed that the canonical sequence of phonemes corresponding to the  
20 typed text was correct. This may not be the case since this assumes that the typed word or words are not misspelled or mis-typed. Therefore, in an alternative embodiment, confusion probabilities may also be used for  
25 typed queries and/or annotations. In other words, equations (4) would be used even where either the annotation or the query or both are text. The confusion probabilities used may try to codify either or both misspellings and mis-typings. As those skilled in the art  
30 will appreciate, the confusion probabilities for mis-typings will depend upon the type of keyboard used. In particular, the confusion probabilities of mis-typing a word will depend upon the layout of the keyboard. For example, if a letter "d" is typed then the keys

005207-102500



surrounding the key for the letter "d" will have high mis-typing probabilities whereas those located further away from the "d" key will have lower mis-typing probabilities. As mentioned above, these mis-typing probabilities may be used together with or replaced by confusion probabilities for the mis-spelling of the words. These mis-spelling probabilities may be determined by analysing typed documents from a large number of different users and monitoring the type of mis-spellings which usually occur. Such mis-spelling probabilities may also take into account transcription errors caused by mis-keying. In such an embodiment, the dynamic programming constraints used should allow for insertions and/or deletions in the typed input. For example, the constraints illustrated in Figure 11 could be used.

A further alternative is where the text is input via a keyboard which assigns more than one character to each key (such as the keyboard of a mobile phone), where the user must repeatedly press each key to cycle through the characters assigned to that key. In such an embodiment, the confusion probabilities would be adjusted so that characters assigned to the same key as the input character would have higher mis-typing confusion probabilities than those associated with the other keys. This is because, as anyone who has used a mobile phone to send a text message will appreciate, mis-typings often occur because the key has not been pressed the correct number of times to input the desired character.

In the first embodiment described above, during the

00507-2056960

dynamic programming algorithm, equation (4) was calculated for each aligned pair of phonemes. In the calculation of equation (4), the annotation phoneme and the query phoneme were compared with each of the phonemes known to the system. As those skilled in the art will appreciate, for a given annotation phoneme and query phoneme pair, many of the probabilities given in equation (4) will be equal to or very close to zero. Therefore, in an alternative embodiment the annotation and query phoneme pair may only be compared with a subset of all the known phonemes, which subset is determined in advance from the confusion statistics. To implement such an embodiment, the annotation phoneme and the query phoneme could be used to address a lookup table which would identify the model phonemes which need to be compared with the annotation and query phonemes using equation (4).

In the above embodiments, the features of the annotation and the query which have been aligned and matched have represented units of speech. As those skilled in the art will appreciate, the above-described technique can be used in other applications where the features of the query and the annotation may be confusable due to the inaccuracies of a recognition system which generated the sequences of features. For example, the above technique could be used in optical character or handwriting recognition systems where there is a likelihood that a recognition system might mistake one input character for another.

A number of embodiments and modifications have been described above. As those skilled in the art will

005507-103500

appreciate, there are many other embodiments and modifications which will be apparent to those skilled in the art.

09695077-102500